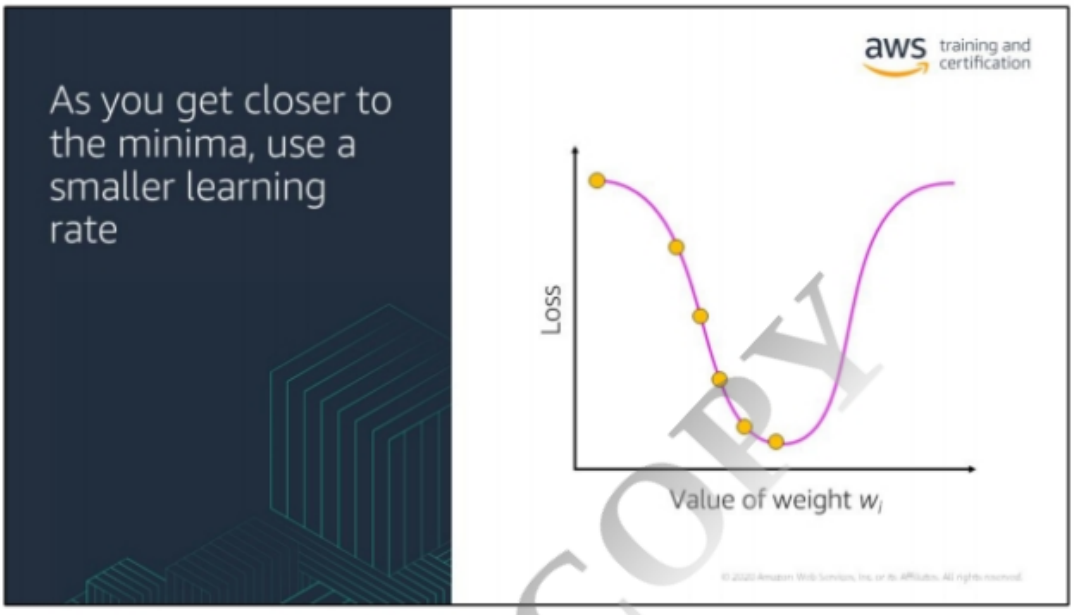
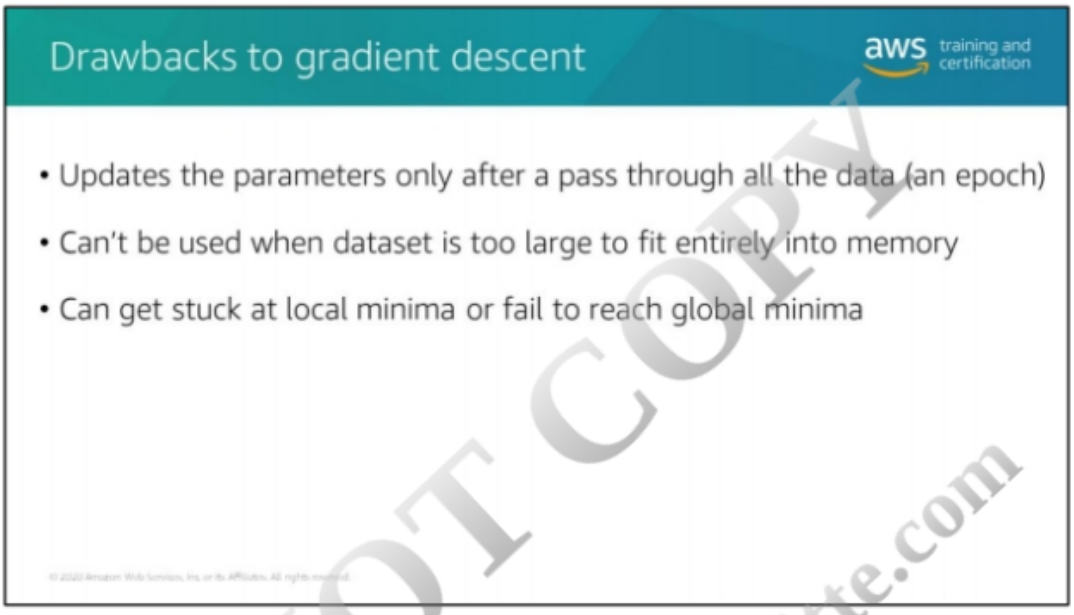
Whereas taking a very small step might cause the model to move very slowly towards the minimum, and fail to reach the minimum within the given time. The step size, also known as "Learning rate" is a hyperparameter to the model.



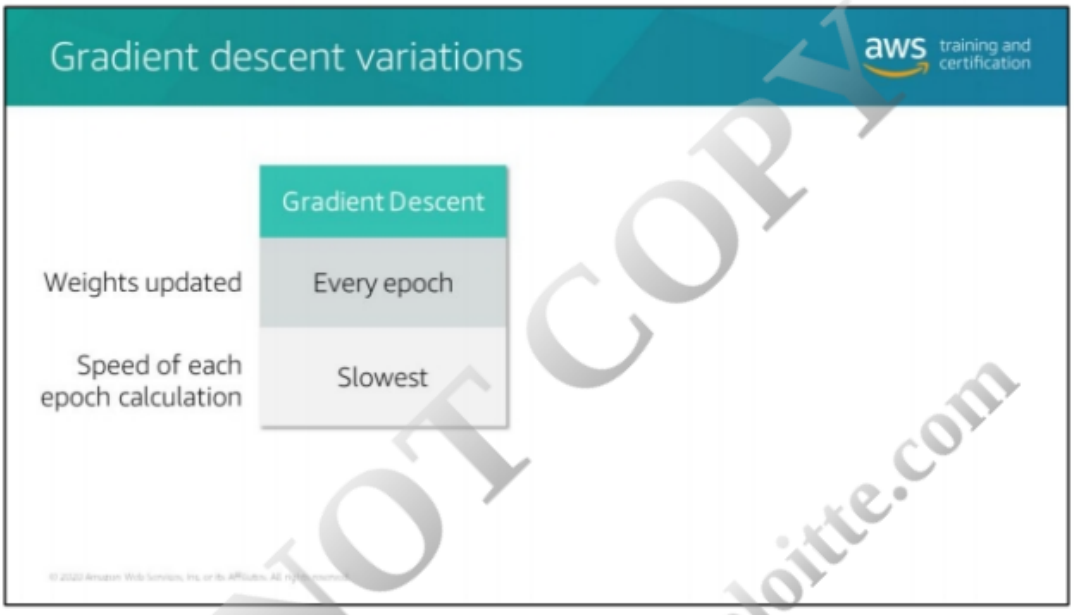
The optimal learning rate depends on how flat or steep the surface of the error function is. In general, a small value of gradient indicates the learning rate could be larger, which means we could safely take a larger step down and not go over the minimum. On the other hand, a large value of gradient indicates the slope is steep and, therefore, we must tread carefully and take small steps, so as not to fall off the cliff.



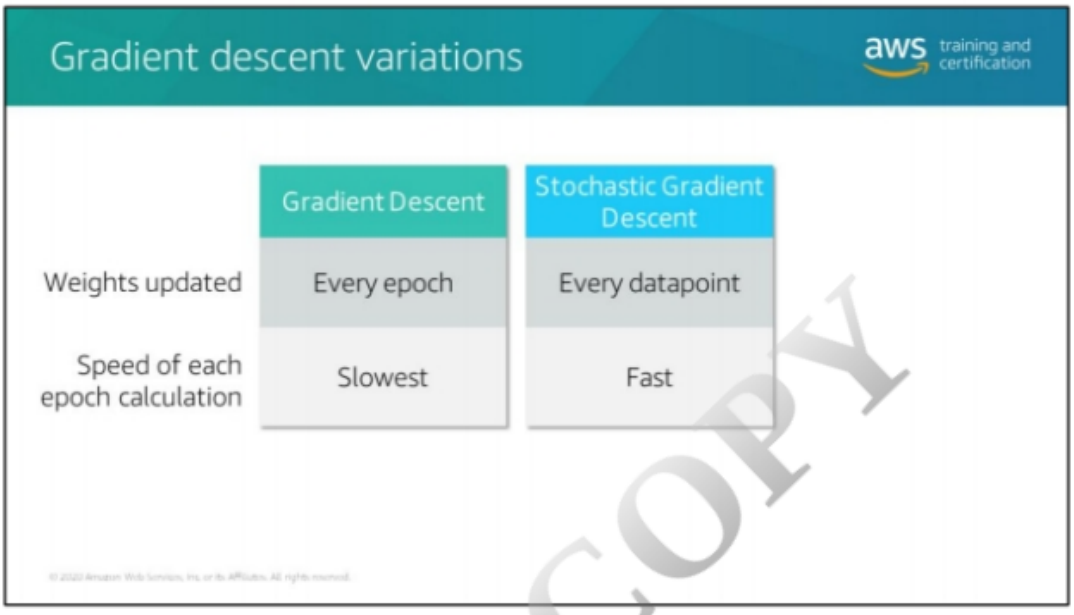
With all the advantages of gradient descent helping the model reach the global minimum, there are some drawbacks as well:
Gradient descent updates the parameters after a pass through the whole data (what is commonly referred to as an *epoch*). As a result, the updates happen very slowly for large datasets and may require a large number of iterations through the whole dataset to converge to the global minimum.

When the dataset is so large that it cannot fit into memory, you cannot use gradient descent.
Gradient descent can get stuck at local minima or fail to reach the global optimum.

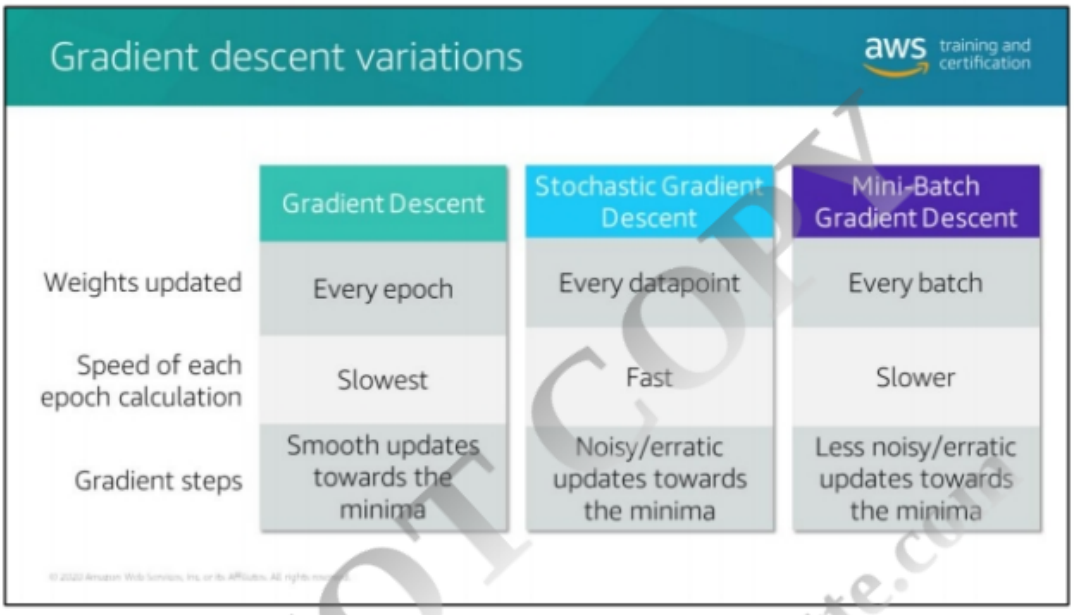
Thankfully, there are ways to mitigate these drawbacks using optimization algorithms that are based on gradient descent like stochastic gradient descent.



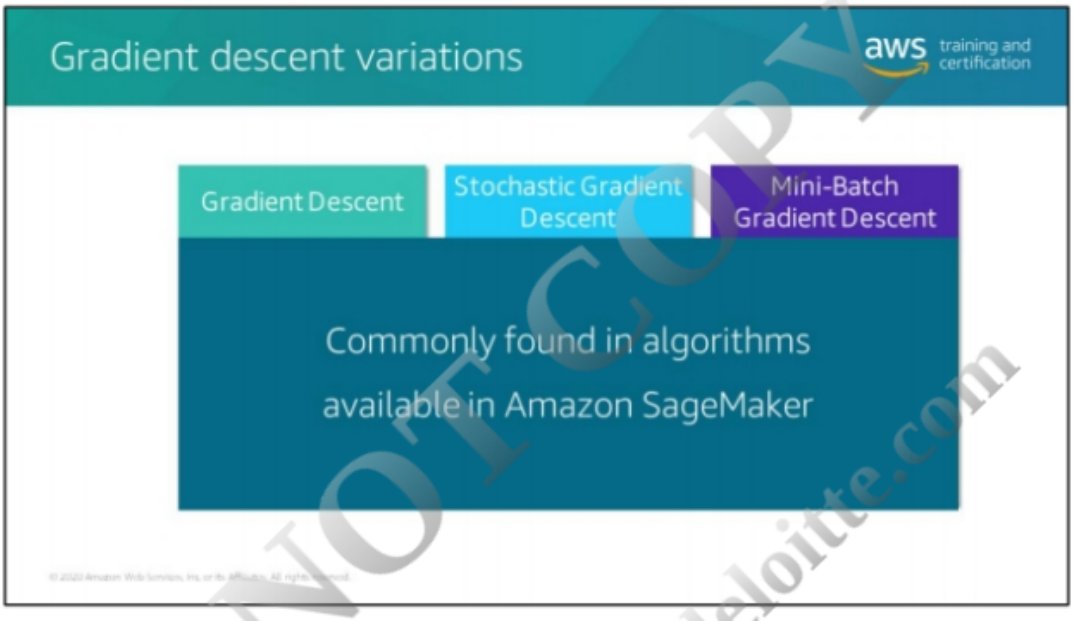
Let's briefly cover some variations to gradient descent. As mentioned in the previous slide, gradient descent only updates weights once it's gone through all of the data, also known as an epoch. Of the three variations we're going to talk about here, gradient descent has the slowest speed to finding the minima as a result, but also has the fewest number of steps to reach the minima.



In stochastic gradient descent, or SGD, you update your weights for each record you have in your dataset. For example, if you have 1,000 data points in your dataset, SGD will update the parameters 1,000 times. With gradient descent, the parameters would be updated only once—in every epoch. SGK leads to more parameter updates and, therefore, the model will get closer to the minima more quickly. One drawback of SGD, however, is that it will oscillate in different directions unlike gradient descent, which will always point towards the minima.

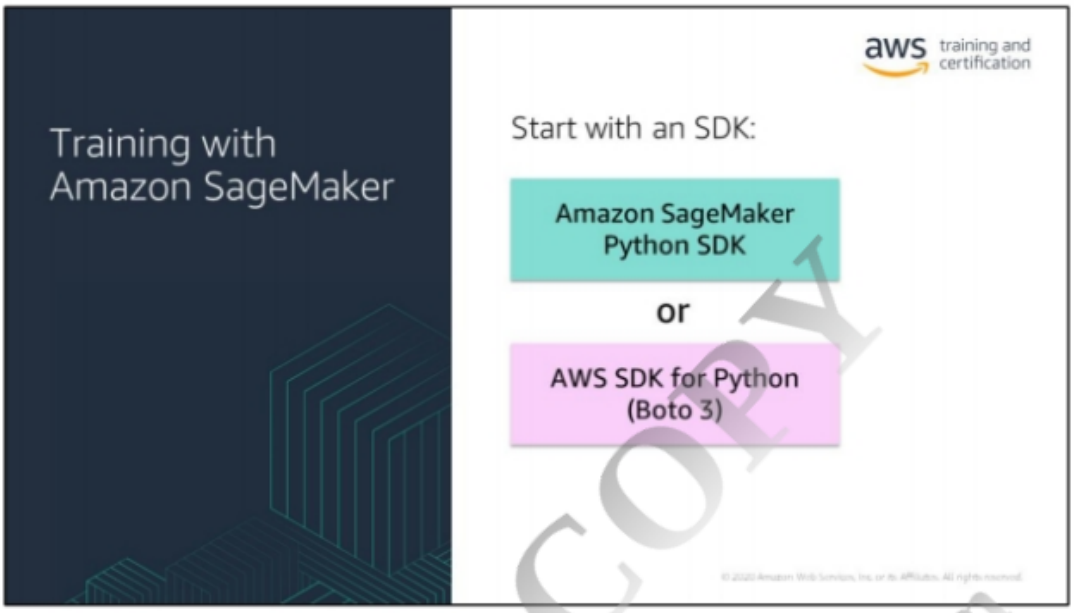


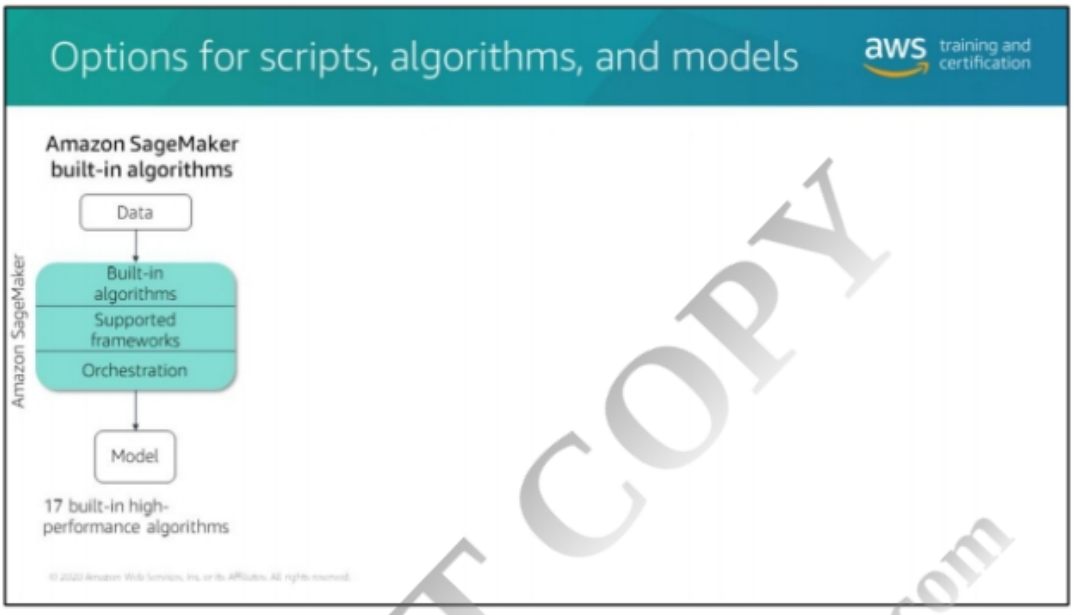
Another technique is to use something that is between gradient descent and SGD called *mini-batch gradient descent*. This approach uses a smaller dataset or a batch of records (also called *batch size*) to update your parameters. Mini-batch gradient descent updates more than gradient descent while having less erratic/noisy updates when compared to SGD. The user defined batch size makes it easy for the user to fit the smaller dataset in memory so that you can make the algorithms run on almost any average computer that a data scientist may be using.



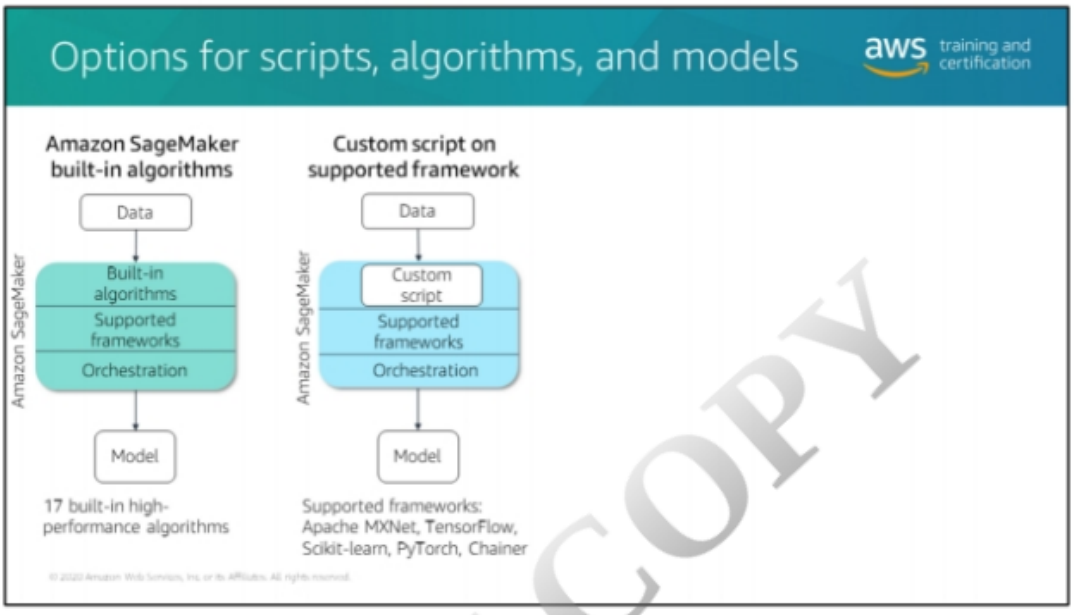
All of these techniques are commonly found in a lot of algorithms as well as in Amazon SageMaker. We'll talk about some of those algorithms and generally how to run training jobs in Amazon SageMaker next.

Now we're ready to begin running a training job.

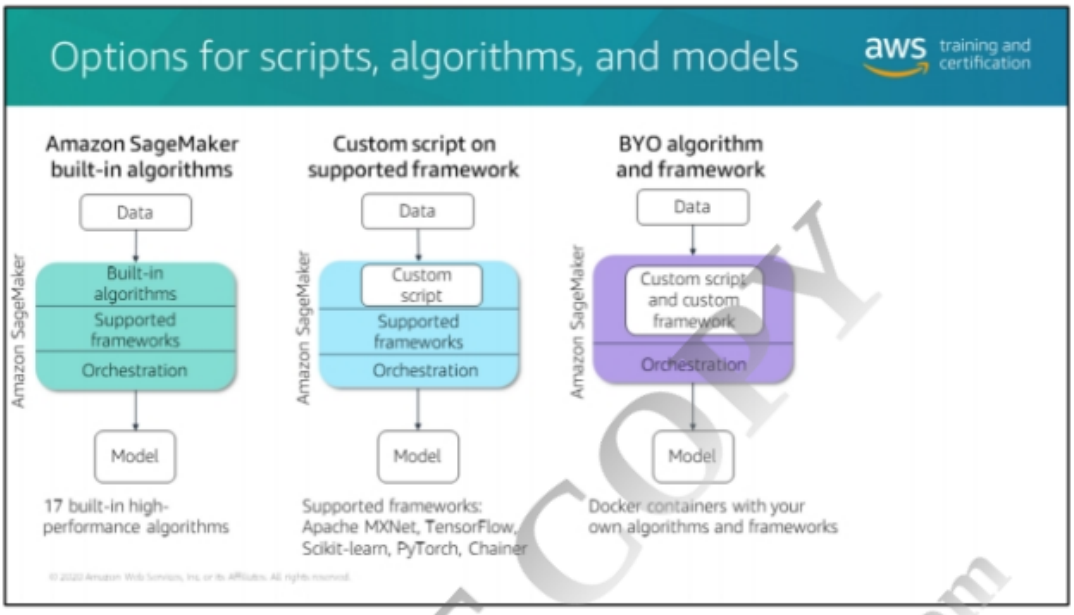




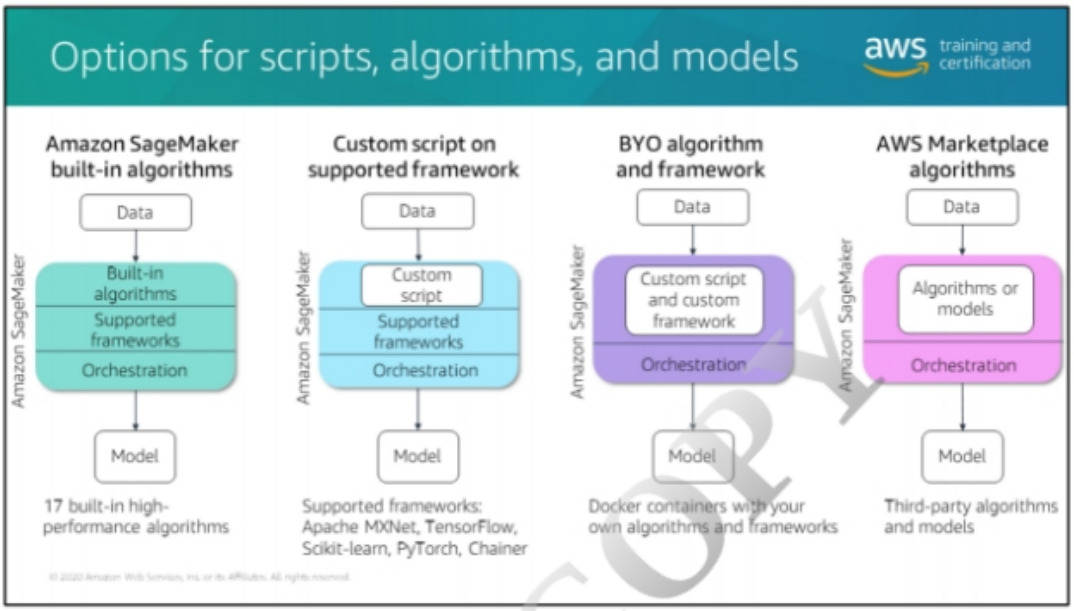
Containers are used behind the scenes when you use one of the Amazon SageMaker built-in algorithms, but you do not deal with them directly. You can train and deploy these algorithms from the Amazon SageMaker console, the AWS Command Line Interface (AWS CLI), a Python notebook, or the Amazon SageMaker Python SDK. The built-in algorithms available are itemized and described in the Use Amazon SageMaker Built-in Algorithms topic.



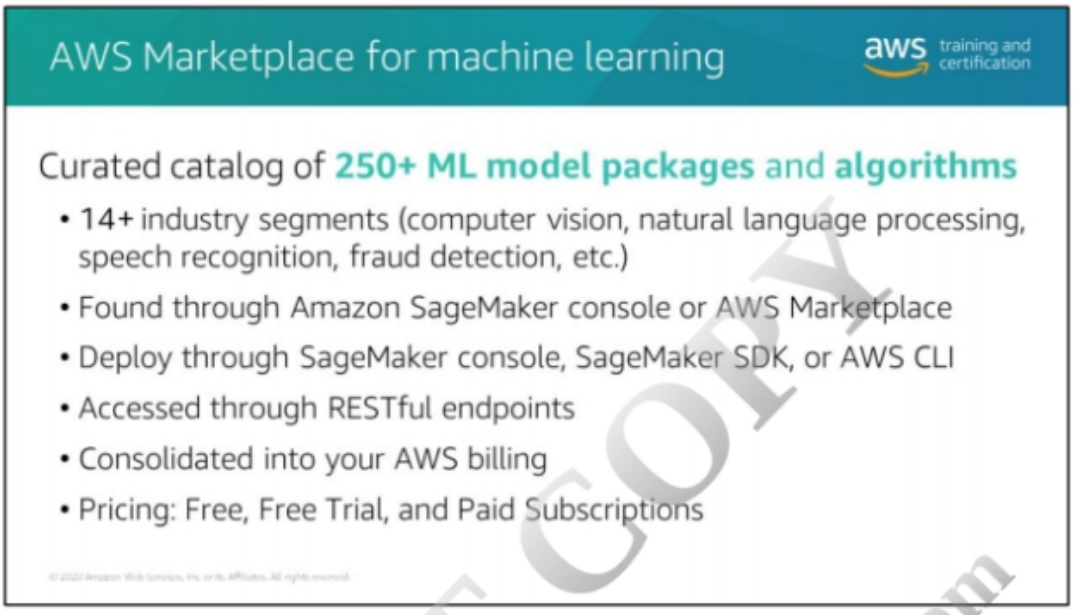
Amazon SageMaker provides pre-built containers to support deep learning frameworks such as Apache MXNet, TensorFlow, PyTorch, and Chainer. It also supports ML libraries such as scikit-learn and SparkML by providing pre-built Docker images. If you use the Amazon SageMaker Python SDK, they are deployed using their respective Amazon SageMaker SDK Estimator class. In this case, you can supply the Python code that implements your algorithm and configure the pre-built image to access your code as an entry point.



If you have additional functional requirements for an algorithm or model that you developed in a framework that a pre-built Amazon SageMaker Docker image doesn't support, you can modify an Amazon SageMaker image to satisfy your needs.



If there is no pre-built Amazon SageMaker container image that you can use or modify for an advanced scenario, you can package your own script or algorithm to use with Amazon SageMaker. You can use any programming language or framework to develop your container. For an example, if your team works and builds ML models in R, you can build your own containers to train and host an algorithm in R as well.

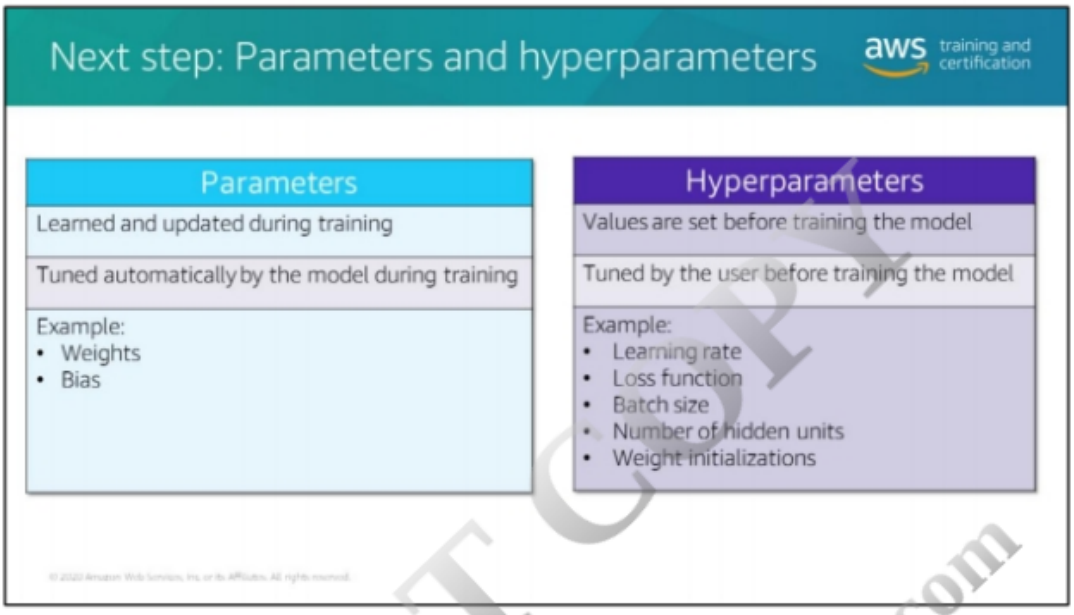


In addition to these scenarios, if you don't want to create your own algorithm but want to buy models and services that were developed by independent software vendors, you can use AWS Marketplace.

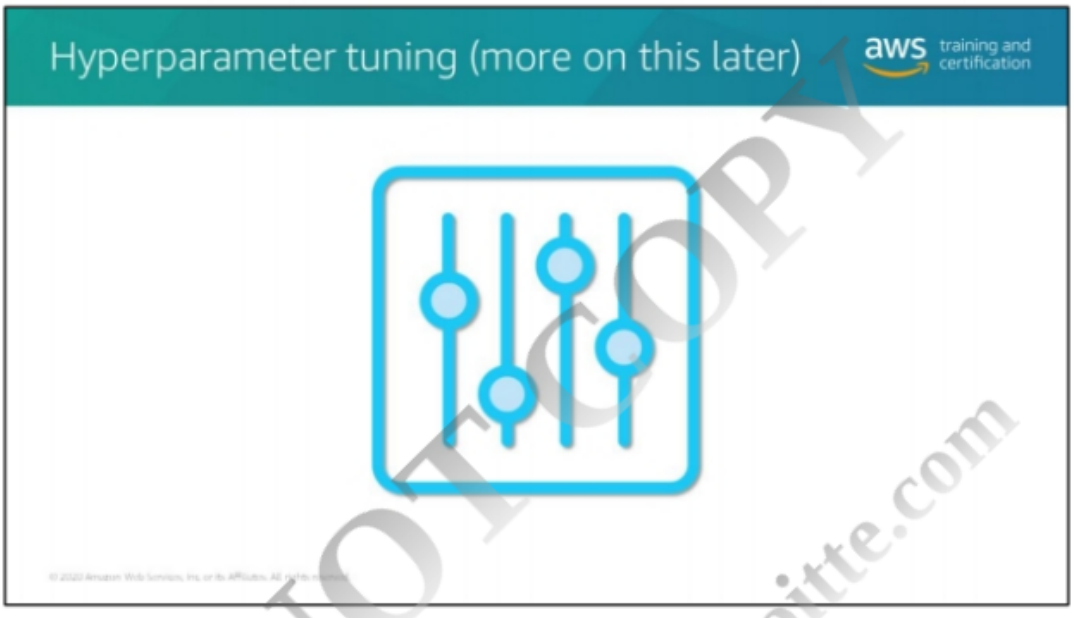
With AWS Marketplace, you can browse and search for hundreds of machine learning algorithms and models in a broad range of categories, such as computer vision, natural language processing, speech recognition, text, data, voice, image, video analysis, fraud detection, predictive analysis, and more.

AWS Marketplace enables sellers to create and provide machine learning algorithms and model packages using Amazon SageMaker. Sellers package their products as Docker containers, upload them to Amazon Elastic Container Registry (Amazon ECR), create the algorithm or model packages in Amazon SageMaker, and add them as free or paid products in AWS Marketplace.

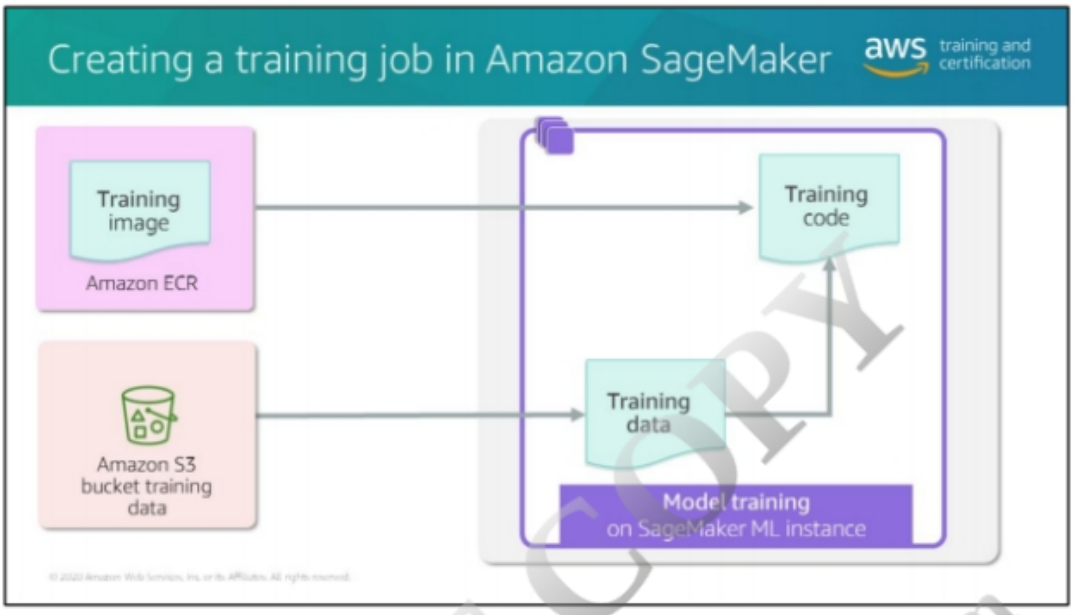
You can find these products through the Amazon SageMaker console or AWS Marketplace, and deploy them on Amazon SageMaker. You can review product descriptions, documentation, customer reviews, pricing, and support information. When you subscribe to an algorithm or model package, the product is added to your list of products on the Amazon SageMaker console. You can also use the Amazon SageMaker SDK, the AWS Command Line Interface (AWS CLI), or the Amazon SageMaker console to create a fully managed inference endpoint. You can access models only through the RESTful endpoints.



Earlier in this module, we talked about weights or parameters, but what are hyperparameters? *Hyperparameters*, compared to parameters, are external to a model and can't be estimated from the data. Hyperparameters are set by humans. Think about hyperparameters as the knobs used to tune the ML algorithm to improve its performance. One example of a hyperparameter is the learning rate. We saw that the learning rate hyperparameter can have an impact on how quickly your model is able to converge to a global minimum.



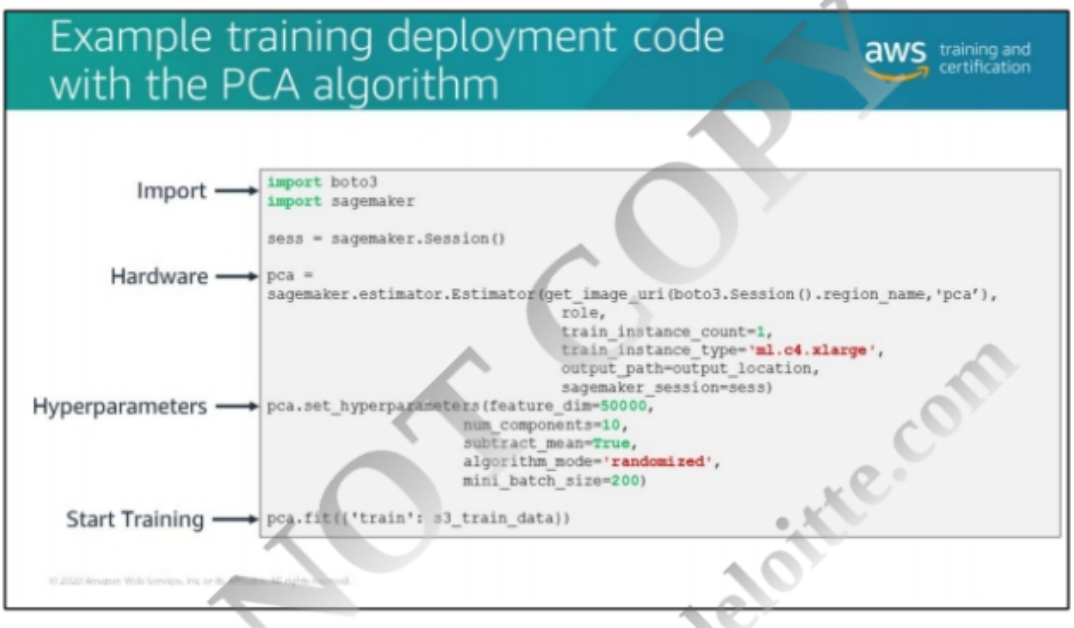
Hyperparameters govern your model's training process. By tuning them, you're tuning the way the algorithm learns. By choosing the appropriate hyperparameters, you can really impact your model's behavior. Each algorithm has its own hyperparameters that can be tuned, including ones that can optimize in order to reduce overfitting and underfitting. The process for searching for 'the best' hyperparameters is called *hyperparameter optimization*. We'll cover that later.



To train a model in Amazon SageMaker, you create a training job. The training job includes the following information:

- The URL of the Amazon S3 bucket where you've stored the training data.
- The compute resources that you want Amazon SageMaker to use for model training. Compute resources are ML compute instances that are managed by Amazon SageMaker. Amazon SageMaker provides a selection of instance types optimized to fit different machine learning (ML) use cases. Instance types comprise varying combinations of CPU, GPU, memory, and networking capacity and give you the flexibility to choose the appropriate mix of resources for building, training, and deploying your ML models. Each instance type includes one or more instance sizes, allowing you to scale your resources to the requirements of your target workload.
- The URL of the S3 bucket where you want to store the output of the job.
- The Amazon Elastic Container Registry path where the training code is stored.

We'll wrap up this module with a demo of how we run a training job in Amazon SageMaker, but first let's talk about another important component of the training process: Hyperparameters.



You can use the Amazon SageMaker SDK and AWS Python SDK boto3 to call your SageMaker training jobs. You can call Estimator, the high level interface for Amazon SageMaker for model training. Estimators make it easy for you to specify the hardware you want for your training job including the container for your model, the training instance count and type of instance to use. Next, you can specify the hyperparameters that you want to use for your training job using the set hyperparameter method.

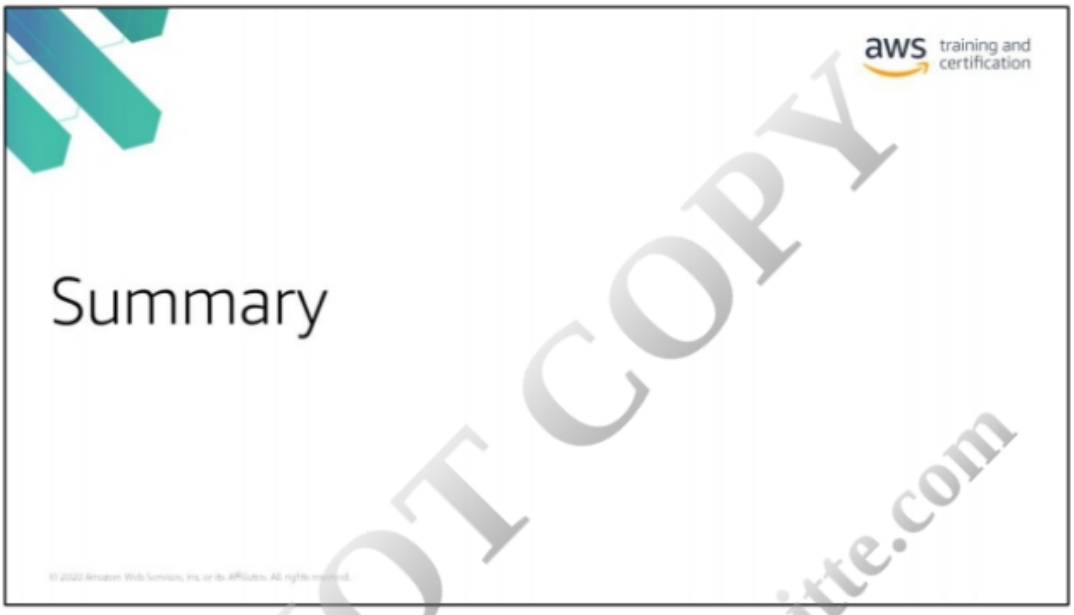
Finally, you call the fit() method to do actual training. The fit() API calls the Amazon SageMaker CreateTrainingJob API to start model training. The API uses configuration you provided to create the estimator and the specified input training data to send the CreatingTrainingJob request to Amazon SageMaker.

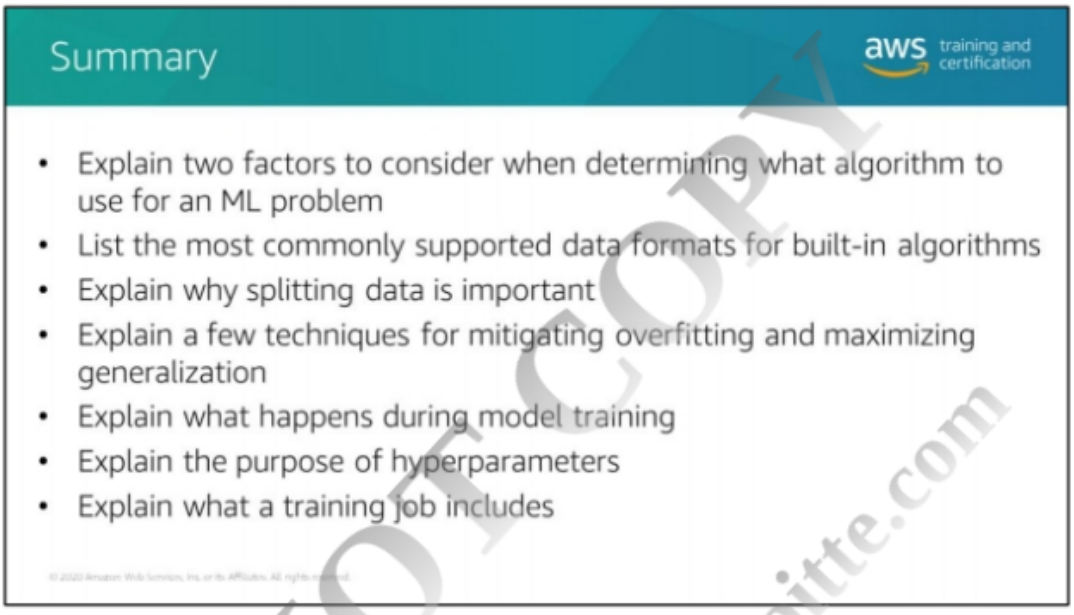
In this demo, we're going to create an Amazon SageMaker training job, and when it's complete we'll dig into the logs to take a deeper look.

Students will get to put what they've learned about Model Training into practice after the following module on Model Evaluation.

Let's wrap this module up by running through a demo of running a training job in Amazon SageMaker.

Note: This demo is for creating a training job, and the lab connected to this demo will take place after the following module on evaluation.





Two factors to consider when determining what algorithm to use for an ML problem:
- Consider what business problem (classification, regression, recommendation | clustering, topic modeling, embeddings, anomaly detection, dimensionality reduction) you have and what data text, speech, image/video, time series) you have.

List the most commonly supported data formats for built-in algorithms
- Comma separated values and Record-IO protobuf

Why splitting data is important:
- Splitting data helps build a model that generalizes well. In other words it works well on both data it already has and the data it does not have. Evaluating a model with the same data that it trained on will lead to *overfitting*.

A few techniques for mitigating overfitting and maximizing generalization:
- Simple Hold-Out Validation - is when you split your data into multiple sets
- K-Fold Validation or cross-validation - methods to compare the performance of multiple models
- Leave-One-Out cross-validation - the K is equal to N. Every time we leave one data point out for testing, we are using the rest in the training data. This is usually used for very small datasets where every data point is very valuable.
- Stratified K-Fold cross-validation - There's also stratified K-Fold cross-validation, which is often used when there are seasonalities or subgroups in small proportion in the data set. Stratified K-Fold cross-validation is going to ensure that for each fold, there are some equal weight proportions of the data for every different fold.

- Iterated K-Fold Validation with Shuffling - applying K-fold validation multiple times, shuffling the data every time before splitting it K ways

What happens during model training:

- During training, the machine learning algorithm updates a set of numbers known as *parameters* or *weights*. The goal is to update the parameters in the model in such a way that the computed or predicted output becomes as close as possible to the true output (as seen in the data).
- Explain the four main scenarios for running scripts, algorithms, and models in the Amazon SageMaker environment.
- Use a built-in algorithm *(previously discussed)*
- Use pre-built container images
- Extend a pre-built container image
- Build your own custom container image

The purpose of hyperparameters:

- Hyperparameters govern your model's training process. By tuning them, you're tuning the way the algorithm learns. By choosing the appropriate hyperparameters, you can really impact your model's behavior. Each algorithm has its own hyperparameters that can be tuned, including ones that can optimize in order to reduce overfitting and underfitting. The process for searching for 'the best' hyperparameters is called *hyperparameter optimization*.

What a training job includes:

- The URL of the Amazon S3 bucket where you've stored the training data.
- The compute resources that you want Amazon SageMaker to use for model training. Compute resources are ML compute instances that are managed by Amazon SageMaker. Amazon SageMaker provides a selection of instance types optimized to fit different machine learning (ML) use cases. Instance types comprise varying combinations of CPU, GPU, memory, and networking capacity and give you the flexibility to choose the appropriate mix of resources for building, training, and deploying your ML models. Each instance type includes one or more instance sizes, allowing you to scale your resources to the requirements of your target workload.
- The URL of the S3 bucket where you want to store the output of the job.
- The Amazon Elastic Container Registry path where the training code is stored.